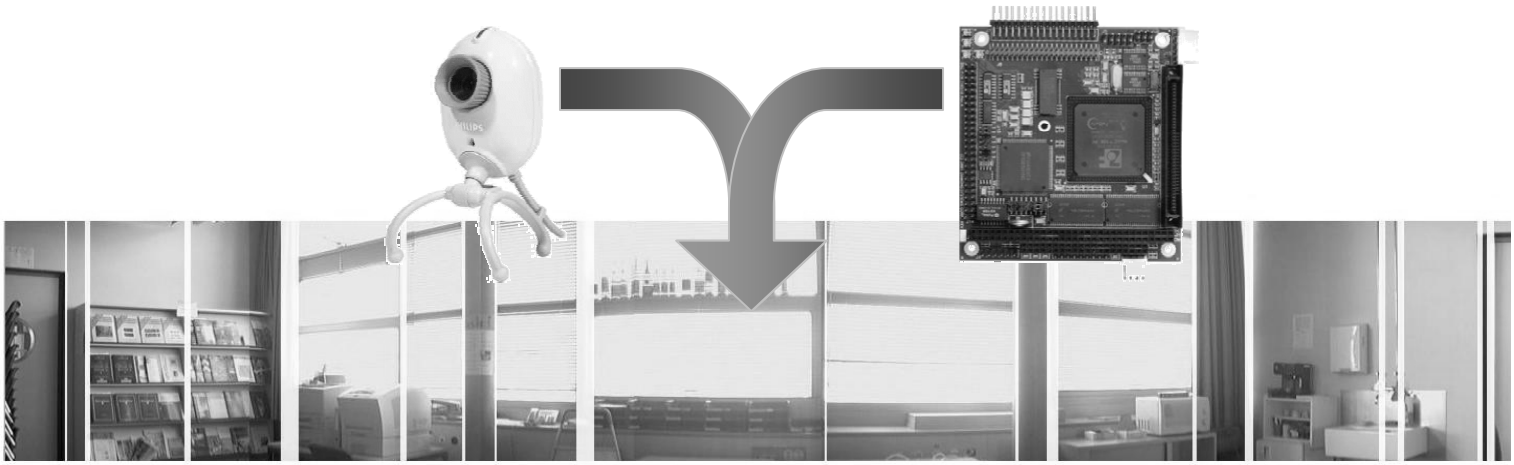
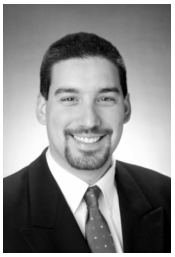
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

EPFL > FaculteSTI > IPR > LSRO > VRAI
Virtual Reality and Active Interfaces Group

PROJET DE SEMESTRE ETE 2003

Système de navigation "Low-cost"





PROFESSEUR : REYMOND CLAVEL
ASSISTANTS : TERRY FONG, CHARLES BAUR
ETUDIANT : ALEC AVEDISYAN

Table des matières

Introduction

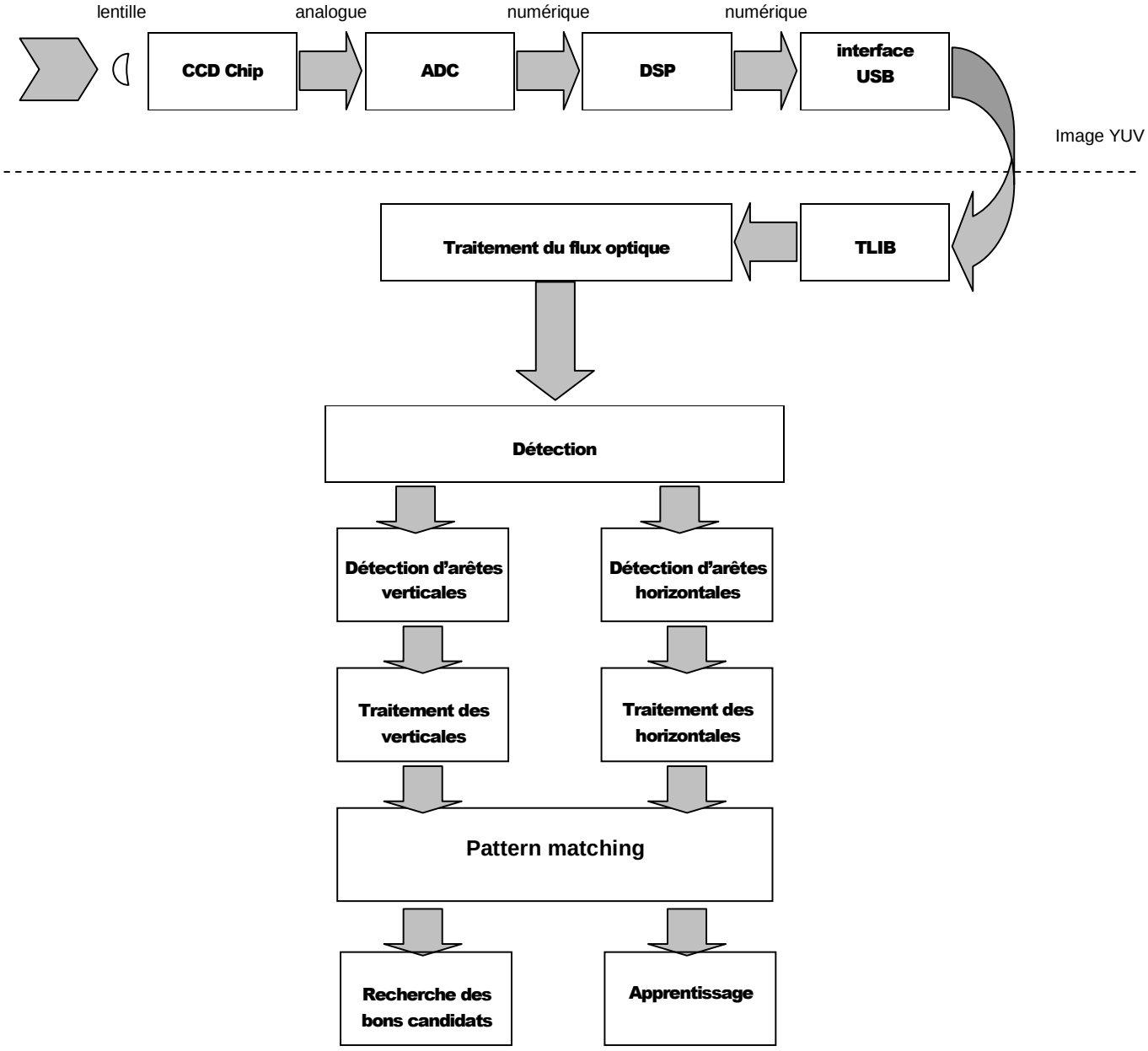
Dans ce projet, plusieurs tâches vont être abordées principalement afin d'améliorer le système d'acquisition "low-cost". Suite à cela, la tâche consistera à développer un système de navigation efficace, qui a pour contrainte principale l'utilisation d'une "webcam" en tant qu'unique capteur et d'un système embarqué qui demande des ressources mémoires faibles et des temps de calcul qui soient adaptés.

Description du projet

Le but de ce projet consiste à implémenter un système de navigation qui soit adapté à un système embarqué low-cost. Ce travail a pour but d'être intégré par la suite dans un projet de navigation pour systèmes embarquables. L'idée de ce projet est qu'à long terme on puisse effectuer plusieurs types d'opérations, en se basant uniquement sur les informations reçues par une caméra. Les fonctions que l'on aimerait par la suite intégrer dans le projet seraient un système de vision pour faire de l'odométrie, un système d'évitement d'obstacles, un système de reconnaissance d'objets, de volumes ou d'espaces, … . Dans notre cas, c'est le dernier des systèmes qui nous intéresse.

Le matériel dans ce projet a aussi pour but d'être "low-cost"; on entend par là qu'il peut y avoir des contraintes au niveau de la qualité des informations et surtout au niveau des temps de calcul. Il faut donc avoir des algorithmes qui soient adaptés aux systèmes embarquables, c'est-à-dire des calculs efficaces.

Ci-dessous, le schéma du système décrit les processus depuis l'acquisition de l'image en passant par le traitement en différentes étapes de l'image.



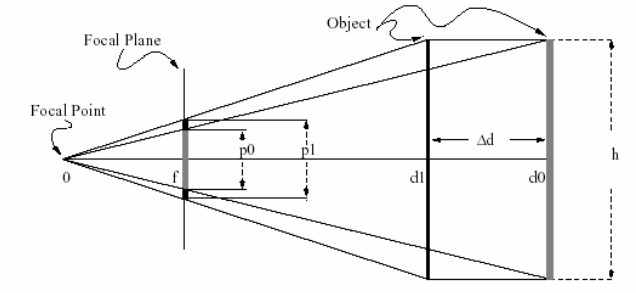
Etat de l'art

Nous allons faire ici un rapide résumé de l'état de l'art en vision pour la navigation, plus particulièrement orienté vers les systèmes monoculaires. Les méthodes d'implémentation et les algorithmes ne seront pas décrits en détail.

Sébastien Dey **[1]** a proposé une méthode de navigation basée sur le flux optique. Ses contraintes sont similaires à celles rencontrées dans le projet que nous allons entreprendre : une seule source de vision pour la navigation d'un robot. Son autre contrainte est que le système doit tourner sur un PC104 en temps réel. Son but n'est pas complètement atteint car le flux optique et le temps de calcul sont trop élevés pour un traitement en temps réel. En effet, il faut plus d'une seconde pour le traitement d'une image.

La méthode de traitement du flux optique est cependant très intéressante. On subdivise l'image en 9 zones de mêmes tailles, pour effectuer des traitements individuels sur chacune des zones. On y applique deux types d'opérateurs d'intérêt, le premier est un opérateur basé sur le gradient, et le second sur le poids du masque. Puis on les teste et enfin on choisit le meilleur des deux opérateurs. Ces calculs sont effectués sur les trois canaux (RGB). On a ainsi 9 vecteurs significatifs qui, selon leurs dispositions, nous donnent le sens du déplacement de l'image. Ces vecteurs peuvent aussi nous informer de l'existence d'un obstacle.

Cette technique reste fortement topologique, afin d'avoir des informations métriques de la position ou de la taille d'un objet, il existe pour cela un algorithme très efficace concernant la vitesse de calcul. Il s'agit de l'algorithme de Looming **[2]**. Cette méthode utilise la distance de déplacement pour calculer la dimension ou la position d'un objet. Elle suppose, bien sûr, de disposer de l'odométrie ou de tout autre technique qui puisse retourner une valeur de déplacement de manière précise.



Ces méthodes susmentionnées ne nous donnent par contre aucune information quand à l'espace dans lequel nous nous trouvons. Pierre Lamon [3] a développé une méthode qui se nomme le « Fingerprint ». Elle consiste à créer l'empreinte visuelle selon un codage prédéfini d'un lieu à partir d'une vision panoramique. L'idée de base est de dire qu'en chaque point d'une salle, le système relève un Fingerprint unique. On considère ici, deux types d'informations :

D'une part, les verticales, qui sont détectées au moyen du gradient de l'image avec la matrice de Sobel [8]

D'autre part, la détection des traces de couleur, qui se fait au moyen d'une conversion de l'image en HSI. Le canal H est extrait puis chaque tranche de couleur est évaluée selon le schéma de vote Fuzzy. A chaque tranche est attribuée une lettre afin de coder le Fingerprint.

Puis cette information est étalonnée avec les Fingerprint existant dans la base de données selon l'algorithme de l'énergie minimum.

Une des principales contraintes, afin d'obtenir un maximum de rendement dans ce système, est de se placer très précisément à l'endroit ou a été capturé le Fingerprint pour le matching. Une autre contrainte est la création d'une image panoramique à partir de fragments précis d'image, cela demande un temps de calcul important pour le matching. C'est pourquoi on préfère utiliser un miroir sphérique (« oeil de poisson ») pour capturer directement l'image panoramique, mais cela, dans notre cas, sort du contexte de dispositif low-cost.

Une autre méthode a aussi été développée sur l'utilisation du miroir sphérique par Niall Winters et José Santos-Victor [4]. Elle consiste à diviser l'image sphérique en 16 parties, puis de leur donner un rang par rapport à l'utilité de l'information qui se trouve sur l'image. Ce n'est pas tant la mise en œuvre technique qui nous intéresse dans ce cas (car elle est lourde), mais plutôt l'idée de ne traiter que le partie la plus intéressante de l'image.

Dans un tout autre domaine, l'INRIA [5] propose une méthode d'indexation d'image et de reconnaissance d'objets très efficace. Cette méthode permet entre autres de trouver des points similaires à une image qui est dans une base de données et qui aurait un angle de vue et une échelle différentes de l'image originale. Cela peut se révéler une méthode très efficace dans le cas de la navigation d'un robot, en effet on peut par cette information se passer d'une odométrie précise.

Dans cette même idée, l'article Jianbo Shi Good Features to Track[6] débat des techniques de base afin d'extraire d'une image les meilleures types d'information (features).

Zachary Dodds et Gregory D. Hager [7] ont aussi écrit un article très intéressant sur la couleur en temps qu'opérateur d'intérêt pour la navigation. Cet article nous intéresse d'autant plus que nous allons aussi utiliser la couleur comme feature de notre système.

Choix et implémentions du système de navigation

Nous rentrons ici dans le vif du sujet:: la recherche d'un système de navigation qui soit efficace et qui prenne en compte les différentes contraintes qui ont été citées.

Description du système choisi

Le choix que nous avons effectué pour l'algorithme de navigation, a été directement en relation avec les contraintes posées. Il nous fallait trouver un système qui puisse extraire d'une image, l'information de manière très efficace. Le temps de calcul est en outre très dépendant de la qualité de l'information qui va être extraite. Un objet extrait de qualité, d'où l'on peut tirer un grand nombre d'informations demandera un temps de calcul important. C'est pourquoi nous avons opté pour l'extraction de "features" dans le flux optique qui soit facilement détectable, tout en extrapolant et en stockant un nombre maximum d'informations possibles sur le milieu dans lequel il se trouve. (Nous utiliserons le mot "features" pour dire type d'information extraite d'une portion ou de la totalité de l'image traitée.)

La principale idée que nous avons eu a été d'implémenter un détecteur de ligne verticale afin de créer en quelque sorte un "spectre" ou un "code génétique" d'un espace dans lequel on se trouve. Cette idée est en partie inspirée du projet "Deriving and matching image fingerprint sequences for mobile robot localization" [3].

Le traitement des ces features dans le cadre de la navigation à été séparée en deux parties principales:

La première phase consiste à créer le spectre de l'espace dans lequel on se trouve en effectuant un tour de la caméra embarquée sur lui-même d'au moins 360° à 720° sans intervention de l'odométrie ou de tout autre capteur externe. Puis, une fois cette opération effectuée, commence la phase de "matching" qui consiste à comparer l'information mesurée à l'information existante qui à été préparée lors de la création du spectre.

Puis une seconde phase, plus complexe, qui effectuera la modélisation de l'espace en 3D selon le mouvement de la caméra. Plusieurs méthodes existent, afin de faire de bonnes estimations quand à la position ou la taille des features de manière sûre. (Cette partie ne sera pas traitée dans le cadre ce projet de semestre, néanmoins quelques suggestions seront proposées)

Afin d'effectuer ces opérations de la meilleure manière, nous devons nous poser un certain nombre d'hypothèses et de contraintes qui sont directement liées au milieu dans lequel la caméra (un robot) va évoluer. Ces contraintes son principalement dues au fait qu'aucun capteur autre que la caméra pour l'extraction de features ne sera utilisé; en outre les différentes contraintes sont celles qui sont propres à l'être humain lorsqu'il ne peut pas faire de différence ou qu'il ne peut porter de jugement. Les contraintes sont les suivantes :

- La hauteur du point de vue de la caméra sera constante. Ne seront acceptables que de faibles variations dues à la rugosité du terrain.

- Le milieu doit rester relativement stable ("immobile") pendant la période d'apprentissage, les mesures sont à éviter, si possible, trop proche d'un obstacle, ce qui créerait une zone d'ombre d'une partie du milieu dans lequel le système évolue.

- L'environnement ne doit pas être trivial. Par exemple, une chambre totalement ronde vide, qui n'aurait ni faces, ni arêtes, et qui, de plus, serait dans une couleur unie, ne permettrait pas au système de se repérer. Il en est de même pour une personne qui ne disposerait pas de capteurs externes (boussole, GPS).

Implémentation et mise en œuvre du système

Détection

Détection des verticales (captureV.cpp)

Il existe plusieurs algorithmes de détection d'arêtes très efficaces (p.ex opérateur de Sobel, mais qui malheureusement demande des temps de calcul trop importants pour que l'on puisse faire une navigation en temps réel). De plus, dans notre cas, nous n'avons été intéressés "que" par les arêtes verticales, et non pas par les autres arêtes qui peuvent créer inutilement du bruit. L'algorithme qui a été implémenté est très simple et a un temps de traitement très faible. En effet, on n'utilise pas de calcul matriciel, mais tout simplement l'intensité de couleur de chacun des vecteurs verticaux qui composent l'image.

$$\Delta j = \sum_{i=0}^{y} pixelIntensity(j+1) - \sum_{i=0}^{y} pixelIntensity(j)$$
$$\Delta j \geq seuil$$
$$\sum_{j=0}^{x} (\Delta j)$$

Equation 1 La complexité est dépendante du nombre de pixel de l'image et est par conséquent de l'ordre de n : $O(n)$

Chacune des valeurs est donc comparée à la suivante. Si le Δj dépasse un certain seuil, on considère qu'il y a à cet endroit une arête verticale.

Diverses optimisations de l'algorithme sont également apportées. Il est par exemple inutile de calculer la valeur pour tous les pixels; on peut se contenter de prendre des pixels à intervalles réguliers de valeur n.

n	Seuil delta	Images / seconde	Résultats visibles
1	2000	18.4	Lent
2	1500	19.6	Lent
3	700	20.2	Bon
5	500	20.7	Excellent
10	300	21.0	Bon
20	150	21.3	Médiocre
40	70	20.6	Mauvais

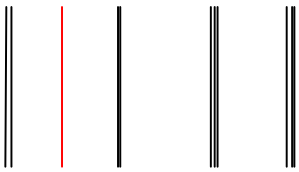
Table 1 Voici une illustration du gain de temps ainsi que de la qualité des résultats. La valeur qui a été retenu pour n=5, Le nombre de pixel à calculer est par conséquent divisée d'un facteur 5.

Figure 1 L'image à gauche avant la détection d'arêtes, et à droite après.

Stabilisation des verticales (stableV.cpp)

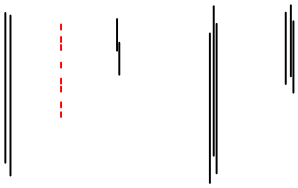
La stabilisation consiste à éliminer le bruit qui crée des arêtes inutilisables, qui par exemple ne seraient pas stables. Elle consiste également à fusionner d'autres arêtes qui, par contre, pourraient être vues comme plusieurs arrêtes distantes.

Ici plusieurs méthodes on été implémentées, mais l'efficacité de certaines d'entre elles n'était pas satisfaisante. Un des algorithmes supposait que pour qu'il y ait une arête, celle-ci devrait avoir une certaine longueur; dans le cas contraire elle serait éliminée. L'information de longueur (début, fin, hauteur), peut être très précieuse pour la reconstitution d'espace. Cette méthode éliminerait, en outre, parfaitement les surfaces tramées, qui sont souvent source de bruit.



Avant traitement des arêtes, l'arête rouge n'est pas stable.

Recherche de la continuité d'une arête avec un seuil minimal de tolérance pour une arête non continue.



Résultat avec calcul des longueurs d'arêtes, élimination des valeurs aberrantes. Moyenne dans le voisinage d'une arête. Pour le calcul de la moyenne, ou effectue une superposition de i images (i=5) puis on compte le nombre d'arêtes adjacentes, on les somme pour obtenir la position médiane.

Cette méthode s'est avérée extrêmement lente pour un gain en terme de qualité d'information qui n'en valait pas la peine. Il a fallu malheureusement renoncer aux informations supplémentaires que nous apporterait cet algorithme.

Un autre algorithme à été implémenté (appliqueMasque()) qui s'est avéré bien plus efficace. L'opération principale consiste à sommer la valeur de présence de chacune des arêtes sur 5 images consécutives. Puis stocker ces informations dans un vecteur de la largeur de l'image, et enfin appliquer des règles (masques) au vecteur.

Voici les trois règles qui sont posées afin d'avoir un masque stable:

- **Toute valeur voisine de 5 (autre que 0) devient un 5**
- **Tout 5 qui entoure un ou deux zéros devient aussi un 5**
- **Toute autre valeur est éliminée**

0000004000	=>	0000000000
0005005000	=>	0005555000
0005525000	=>	0005555000
0000550000	=>	0000550000
0005000400	=>	0005000000
0001005000	=>	0000005000
0011253110	=>	0000555000 (Cas extrême)

Table 2 Illustration par des exemples

La complexité de l'algorithme reste ici aussi linéaire $O(n)$. Et on peut aussi remarquer sur le résultat donné par l'analyse de figure 2, que l'algorithme est déjà très efficace pour une perte de performance minime.

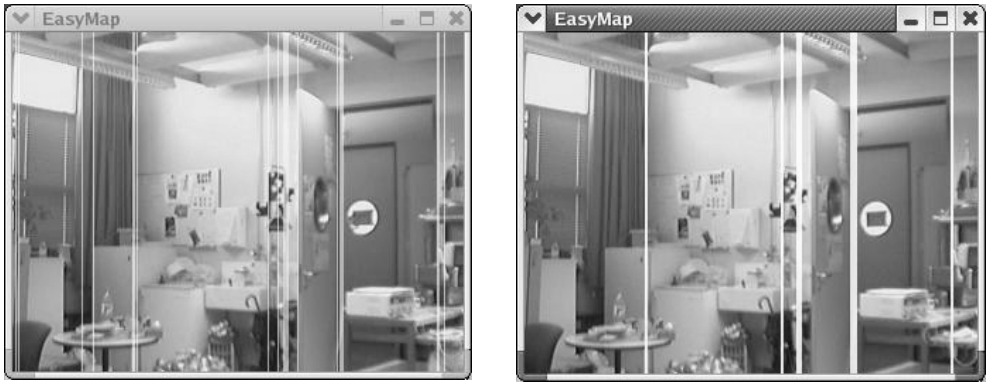


Figure 2 A droite on voit l'image après application des deux masques.

Un second masque est appliqué au premier (appliqueMasque2()) qui discrimine les arêtes de poids = 1 s'il y a déjà des arêtes de poids supérieurs. Les arêtes de poids 1 sont souvent dues au bruit, donc nous les garderons uniquement dans le cas où il n'existe pas d'arête de poids supérieur. Ce cas se présente, par exemple, quand la caméra pointe sur le croisement entre deux parois d'une salle de même couleur, et donc avec un contraste faible pour faire ressortir le coin.

Poids 3	Distance 92	Position 90
Poids 2	Distance 13	Position 182
Poids 5	Distance 34	Position 195
Poids 14	Distance 74	Position 229
Poids 3	Distance 17	Position 303

Table 3 Voici le type d'information qui est généré à la suite de l'application des deux masques

Ces valeurs indiquent le nombre d'arêtes principales qui ont été retenues par prise de vue. Le *poids* correspond à la largeur en arêtes après le traitement des règles. La *distance* indique la séparation entre 2 arêtes, l'information se trouvant toujours sur l'arête de gauche. Et, enfin, la *position* donne la cordonnée sur l'image présente; c'est elle qui permet de calculer la distance entre les deux arêtes.

Détection des horizontales (caputreH.cpp)

Chacune des arêtes aura, comme information supplémentaire, l'état de ses horizontales quand celle-ci se trouve au centre du champ de vision. C'est une information complémentaire qui pourra départager la compétition entre deux arêtes. La valeur des horizontales a, comme très bonne propriété, de nous dire si le point de vision se trouve bien face a une surface ou non. En effet le maximum d'horizontales ne s'affiche que lorsqu'on se trouve perpendiculairement à une surface.

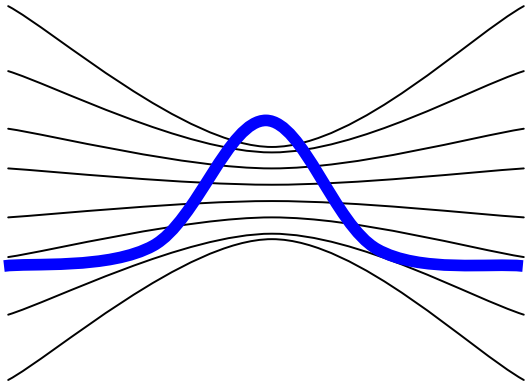


Figure 3 La courbe bleu montre la probabilité de trouver des horizontales par rapport au point de vue.

De plus cette information a comme avantage d'être extrêmement stable. Les problèmes liés à la luminosité ou à la variation de distance affectent peu les mesures.

Figure 4 Exemple de détection d'horizontale

Choix, et problèmes relatifs à l'espace couleur (RGBtoHSL.cpp)

Toute la problématique est de savoir quelle quantité d'information il faudra stocker pour avoir la meilleure estimation de la couleur. Si nous décidons de travailler dans l'espace de couleur RGB, il n'y aura pas de transformation sur la valeur des pixels donc une utilisation directe des données brutes. Deux problèmes se posent directement à nous: d'une part la stabilité de RGB, nous savons que la variation de l'intensité de la lumière fait défaut et a un impact direct sur les trois valeurs RGB. D'autre part, un problème vient de la quantité de données à stocker. En effet, dans cette espace couleur, il nous faut au moins garder les 3 paramètres, voire calculer un $4^{ème}$ dans lequel on stockerait l'intensité de la lumière. Donc il nous faut indiscutablement se résoudre à travailler dans un espace de couleur différent.

Donc, dans le souci principal de réduire la quantité de données à stocker, nous avons opté pour un espace pour lequel 2 valeurs suffisent pour identifier de manière assez stable une couleur. Dans le cas de NRG, les rapport NR et NG sont précis, mais ils ne nous donnent aucunement une valeur d'intensité de la lumière. Des tests ont quand même été effectués pour vérifier ces suppositions, et ces dernières se sont avérées justes. De plus, il est plus difficile, dans cet espace, de pouvoir se représenter mentalement les couleurs.

Dans le cas de HSI ou HSL, par contre, H nous donne de manière très précise et stable la couleur; elle ne dépend en rien de l'intensité de la lumière. Par contre, seul I nous donne une valeur sur l'intensité de la lumière; cette valeur est très importante car sans elle il n'est pas possible de traiter les niveaux de gris. Donc, dans ce cas aussi, deux valeurs sont requises. Contrairement à NRG, l'information contenue dans les valeurs H et I ou L est, quant à elle, bien plus indispensable.

L'avantage de L (Luminance) par rapport à I (Intensité) est qu'elle nous donne de manière plus précise l'information qui concerne le fait qu'une surface soit chromatique ou achromatique. Sinon, pour ce qui est de la vitesse de calcul, les deux sont très proches. ($O(n)$). Le calcul de H se fait selon un algorithme simplifié qui est différent de la méthode traditionnelle, qui demanderait un temps de calcul important.

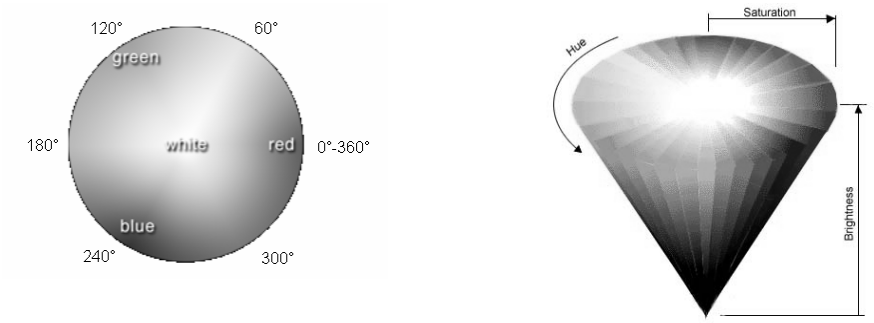


Figure 5 Illustration de l'espace HSL

Un essai a aussi été effectué afin d'optimiser la place de stockage des informations. En effet, nous avons fait une proposition de fusion pour réduire à une seule variable l'information relative à H et I. Le problème maintenant est de pouvoir faire la différence entre H et I. Une des solutions est de garder H comme ayant une valeur allant de 0-360 et I de 1000 à 1100 (1000 étant le noir et 1100 le blanc). L'idée étant de considérer qu'il y a des arêtes qui sont chromatiques et d'autres qui se décrivent par leurs niveaux de gris.

Dans ce genre de solution, d'importants problèmes de singularités se posent : p.ex à éclairage différent un vert foncé avec une lumière claire apparaîtra comme un ton de vert, par contre si la lumière est faible on percevra cette arrête comme un ton de gris.

Cette solution aurait pu, en effet, bien fonctionner, mais il nous faudrait poser comme contrainte, ce qui est à notre sens trop restrictif, une constance dans l'intensité de la lumière.

Aussitôt que nous avons opéré le choix de l'espace couleur, d'autre types de problèmes nous sont apparus, mais cette fois-ci directement liés au matériel. En effet, la webcam dispose d'un système de réglage automatique du gain, et de l'intensité de la lumière. Si l'on veut utiliser les couleurs comme outil principal pour le Matching, il faut absolument stabiliser ces réglages.

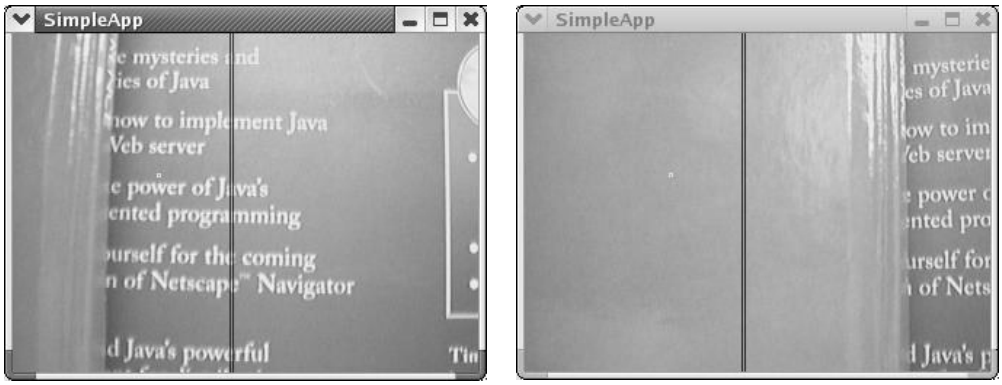


Figure 6 Ici on a affaire à des objets qui ont, dans la réalité, des couleurs vives (vert et bleu). Les deux image montrent la même scène, mais une fois avec le livre bleu qui couvre la majorité de la surface de l'image, et inversement. Le changement de la saturation est flagrant.

En effet, si l'on n'effectue pas cette opération, on obtient des différences de couleurs très importantes: On distingue clairement, sur les images suivantes, que l'auto-régalage fausse complètement les couleurs. Et, à l'extrême, si on place une face unicolore, il n'est plus possible de clairement distinguer les couleurs.

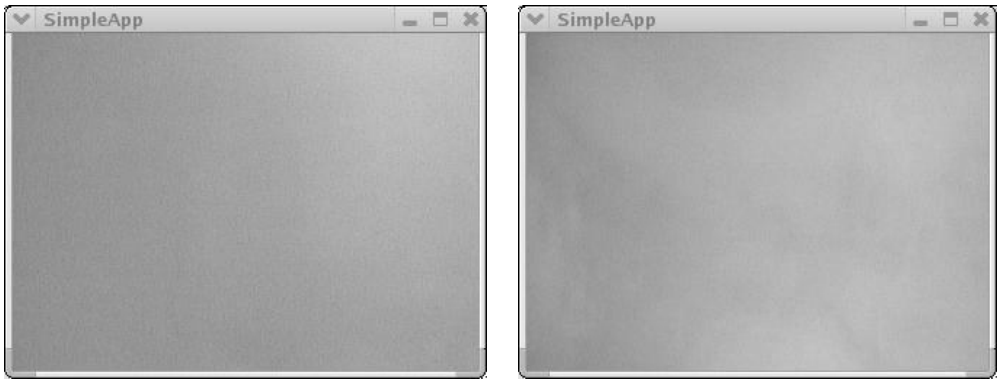


Figure 7 Nous traitons ici les mêmes objets que pour la figure 5, avec à gauche l'objet bleu, puis vert pour la droite.

On peut cependant garder le réglage d'intensité automatique, pour les changements de lumière. Comme l'espace des couleurs est en HSL, l'intensité de lumière ne fait que peu fluctuer les résultats. En revanche, il a fallu stabiliser les valeurs suivantes. Il a fallu fixer les valeurs dans tlDigitizer_v4l.cpp :

```
/*
 * Set manual white balance
 */
wb.mode = PWC_WB_MANUAL;
wb.manual_red = 21000; // (0 - 65535)
wb.manual_blue = 15000; // (0 – 65535)
```

plus d'info : http://www.smcc.demon.nl/webcam/api.html

Ecriture du code génétique (genErator.cpp)

Nous appellerons ici code génétique ou spectre de la salle, la séquence qui contient les informations sur les arêtes. Le code génétique est considéré comme un ruban d'information cyclique, dont on ne connaît pas la longueur.

Voici la description des features qui seront extraites de chacune des arêtes définitives (après application des masques.)

Poids	Largeur de l'arête
Distance	Distance à droite jusqu'à la prochaine arête (axe-axe)
Horizontale	Quantité d'arêtes horizontales mesurée
Position relative	Position relative de chacune des arêtes. (cette information est redondante avec le paramètre de distance. Mais peut être utilisée par la suite)
Teinte (Hue)	La teinte sera mesurée à gauche, au centre, et à droite de l'arête
Intensité	L'intensité sera mesurée à gauche, au centre, et à droite de l'arête

Au total 10 informations seront stockées pour chacune des arêtes.

Matching

Le matching consiste à faire une comparaison entre les informations qui on été stockées dans une base de donnée avec le flux d'informations qui est détecté en temps réel. Il y a, là aussi, plusieurs méthodes afin d'avoir un matching de qualité. Le type d'algorithme qui va être choisi dépend directement des propriétés des features. Nous savons d'ores et déjà qu'il va nous falloir un système très souple qui puisse s'adapter à une quantité importante d'information tout en laissant une marge de tolérance importante. Nous somme aussi conscients que la majeure partie du temps de calcul sera allouée à l'algorithme de matching. Nous savons aussi que la complexité sera d'ordre $O(n^2)$ s'il n'y pas d'optimisation apportée. Cela dit, nous pouvons, avec quelque méthode de discrimination, descendre à $O(\log(n))$

Fonction de fitness (matching.cpp)

La fonction de fitness est la pondération qui va être attribuée à chacun des paramètres selon son ordre d'importance, la qualité ou la stabilité qu'elle peut avoir. La couleur est par exemple beaucoup plus stable que la distance, qui peut facilement être modifiée si le point de vue change. Le travail le plus dur est de pouvoir attribuer le meilleur pourcentage, afin que la réponse soit de bonne qualité.

Paramètre	**poids %**
0 Poids	25
1 Distance	15
2 Horizontal	20
3 posX	0
4 Couleur extraction du H de HSL Gauche	5
5 Couleur extraction du H de HSL Centre	10
6 Couleur extraction du H de HSL Droite	5
7 Intensité extraction du I de HSL Gauche	5
8 Intensité extraction du I de HSL Centre	10
9 Intensité extraction du I de HSL Droite	5

Après un grand nombre de tests, il s'est avéré que les poids suivants sont d'assez bonne qualité. Mais ces valeurs restent ouvertes à des modifications (améliorations). En particulier, en utilisant des algorithmes génétiques et leur propriété. En faisant varier la fonction de fitness, et en croisant les meilleurs candidats (crossover, mutation), la sélection nous fera converger assez facilement vers des valeurs optimales.

Ces valeurs une fois posées nous permettent maintenant de faire des comparaisons avec toutes les arêtes. Comme nous l'avons dit plus haut, la complexité de cette opération est de $O(n^2)$. En effet, toutes les valeurs de la base de données sont comparées, avec les pondérations (ci-dessus), aux valeurs mesurées en temps réel. Le problème maintenant réside dans le fait de traiter au mieux cette importante masse de résultats, afin d'en extraire les meilleurs candidats, c'est à dire les bonnes arêtes. C'est pourquoi il a fallut développer un algorithme de recherche de séquence optimale. Car le problème en soit n'est pas trivial.

Le but de cet algorithme est de trouver une séquence croissante qui se trouve dans une matrice. La difficulté réside dans le fait que l'on ne connaît pas ni le point de départ dans la matrice ni sont point d'arrivé, ni non plus "la pente" quelle aura. Des modifications de pente sont observées quand une arête mesurée n'existe pas dans le base de données ou inversement une arête qui ne serait pas mesurée en temps réel et qui pourtant serait existante dans la base de données.

La méthode la plus simple serait en effet de rechercher toutes les séquences possibles et existantes dans la matrice. Le seul problème de cette méthode est bien entendu sa complexité qui est cette fois-ci de $O(n!)$, et par conséquent inacceptable dans notre cas.

Algorithme 1 : Recherche d'une séquence (indépendant de la position)

Cet algorithme a pour but de trouver la séquence la plus longue, qui soit croissante et qui soit uniquement basée sur les valeurs de fitness maximum.

Le système procède de la manière suivante; on considère que la matrice des fitness ($M_{fitness}$) est déjà calculée et on lui extrait les valeurs maximum pour chacune des lignes. On garde dans un vecteur ces valeurs ($\vec{v}_{valeurs}$), puis dans un second vecteur les valeurs des colonnes correspondantes ($\vec{v}_{colonnes}$). Ce sera cette dernière qui sera traitée et d'où l'on va extraire la séquence optimale.

$$\vec{v}_{colonnes} = \sum_{i=1}^{n} v_i$$

On va parcourir le vecteur $\vec{v}_{colonnes}$ en prenant, à chaque itération, comme référence de départ (a) la valeur présente. Puis nous allons rechercher dans le reste du vecteur à partir de b la valeur minimale qui a comme condition d'être supérieure à la valeur de départ.

$$b = a + 1$$

$$\vec{v}_a \leq \min(\sum_{i=b}^{n} v_i)$$

$$\vec{v}_b \leq \min(\sum_{i=b+1}^{n} v_i)$$

L'avantage est que l'algorithme discrimine les informations aberrantes, et crée des séquences souvent satisfaisantes. Par contre, comme il n'est basé que sur les valeurs maximales de $M_{fitness}$, on arrive souvent dans le cas où une pseudo-séquence est crée à partir de bouts de séquences éparpillés dans la matrice (toujours de façons croissante).

Pour illustrer, nous allons juste donner un exemple d'une bonne séquence et d'une mauvaise. Nous appellerons l'algorithme $g(\vec{v})$ avec un $a=1$.

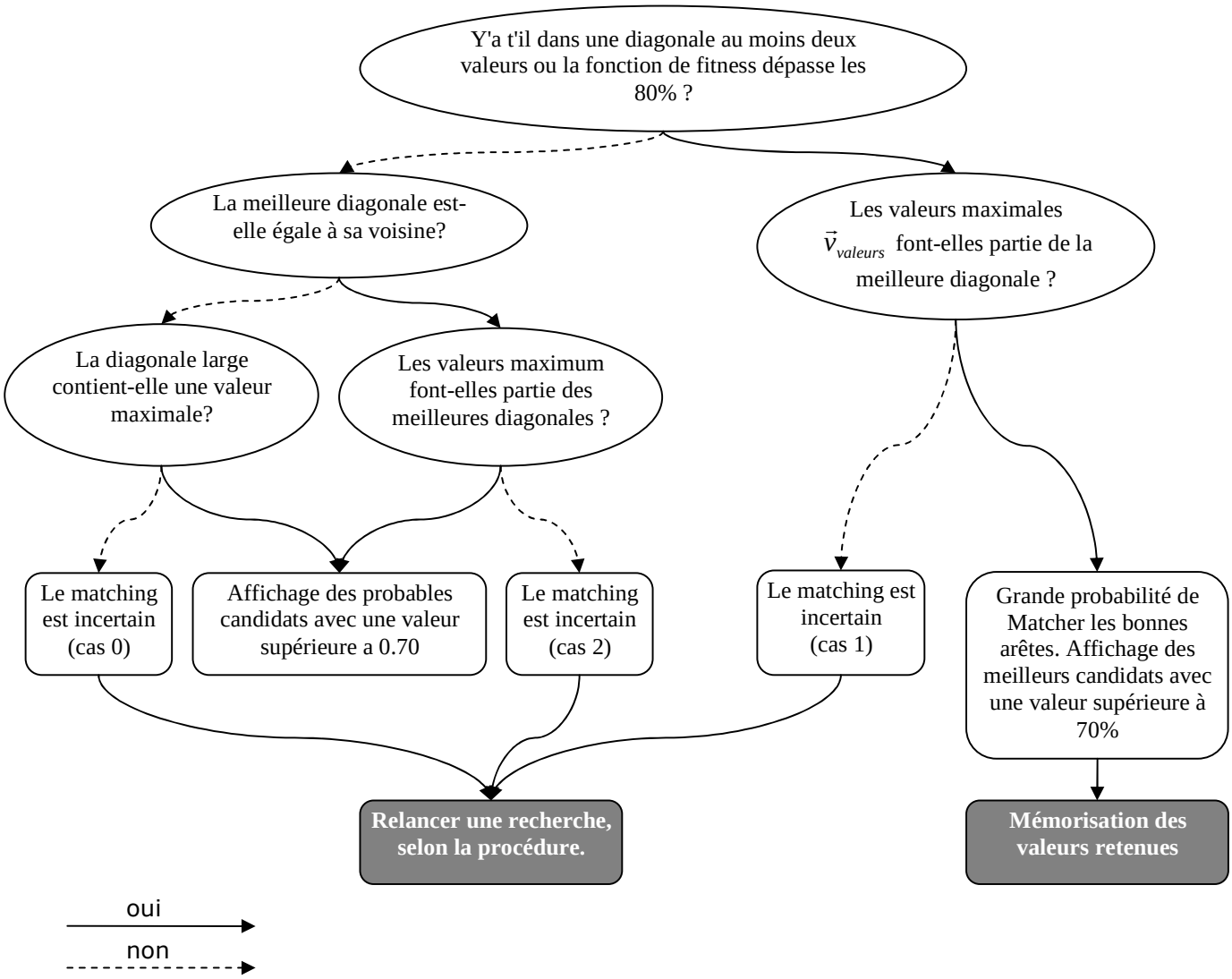
$$\vec{v}_{colonnes} = \{1 \quad 3 \quad 9 \quad 3 \quad 4 \quad 5 \quad 0 \quad 7 \quad 8 \quad 1\};$$
$$g(\vec{v}_{colonnes}) = \{1 \quad 3 \quad 4 \quad 5 \quad 7 \quad 8\}$$

$$\vec{v}_{colonnes} = \{1 \quad 3 \quad 9 \quad 3 \quad 4 \quad 5 \quad 0 \quad 7 \quad 2 \quad 1\};$$
$$g(\vec{v}_{colonnes}) = \{1 \quad 2\}$$

Ce genre de résultats peut être très pénalisant et fausse le matching. C'est pourquoi il nous fallait trouver un second algorithme qui peut être complémentaire au premier, mais qui ne prend pas en compte que les valeurs maximales, mais plutôt une zone de valeurs qui soient supérieures au reste des valeurs de la matrice.

Algorithme 2 : Recherche d'une zone de valeur (dépendant de la position)

Le meilleur moyen de calculer la valeur d'une zone croissante dans une matrice est de faire tout simplement la somme des valeurs sur sa diagonale. Plus la valeur de la diagonale est élevée et plus la probabilité d'y trouver une partie de la bonne séquence est grande. Afin aussi de palier au fait que la diagonale peut ne pas être continûment croissante, on calcule aussi la valeur des diagonales du voisinage. Ce qui nous donne maintenant clairement une zone. Cette méthode est beaucoup plus souple et ne se base pas uniquement sur les valeurs maximales. Puis plusieurs règles sont appliquées dans le but de valider la séquence. Le diagramme de décision binaire décrit les étapes.



Visualisation des résultats

Un système de visualisation à aussi été mis en place afin mieux comprendre dans quel direction de la salle pointe la caméra et quelles sont les arêtes qui ont été reconnues.

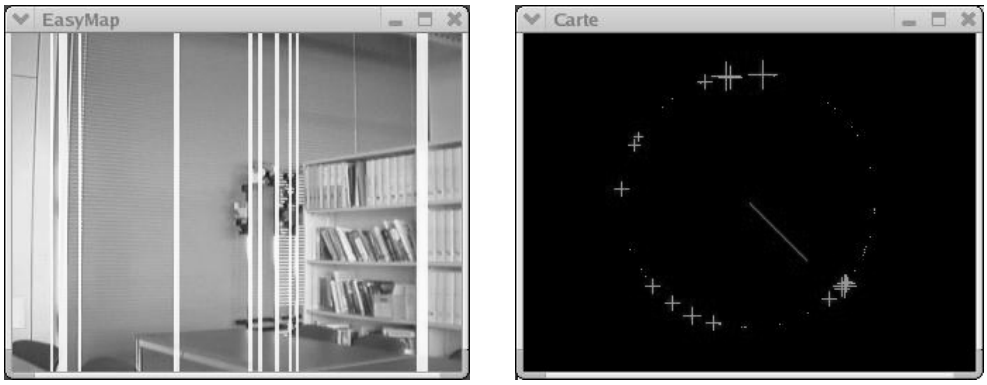


Figure 8 Ce système visuel permet de se repérer de manière plus intuitive. Toutes les arêtes existantes dans la carte génétique de la salle sont indiquées par des croix. Plus les croix sont grandes et plus la probabilité de se trouver face aux bonnes arêtes est bonne.

Amélioration & Développement futur

Il est clair que l'implémentation d'une méthode de navigation basée uniquement sur une seule caméra sur un système embarqué est un défi très passionnant. Il faudrait, pour pouvoir développer une méthode complète, bien plus de temps que celui qui est imparti pour un projet de semestre. Dans cette section, nous allons développer les idées principales qui nous ont motivées et qui ont été le fil conducteur pour la création d'un système de navigation complet.

Apprentissage

La partie apprentissage n'a pas été implémentée, car elle demande pour cela l'utilisation de l'algorithme de matching. L'idée fondamentale derrière la réalisation du matching est que le robot tourne sur lui-même en n'effectuant des sauts qui ne soient pas forcément réguliers. Le robot dicte lui-même la direction dans laquelle regarder, selon qu'une face soit intéressante ou non (jugement par rapport à la quantité d'arêtes mesurées). Puis effectue un matching pour savoir s'il connaît déjà une partie de la séquence ou non.

Afin d'avoir une image aussi stable que possible, il serait préférable de faire l'acquisition non pas sur 5 images comme on le fait pour le matching, mais sur 30 images.

Une idée d'algorithme est proposée.

- Afin d'avoir un départ de bonne qualité dans la code génétique, le robot doit faire une recherche de la face principale; c'est celle qui contient le maximum d'horizontales.

- Puis effectuer des fractions de tour, en gardant tout le temps la même direction.

- Effectuer une acquisition, puis un matching pour voir si la face a des points communs avec une partie existante. Si oui ajout de la nouvelle partie, sinon retour dans la direction inverse.

- L'apprentissage ne sera terminé que lorsque le robot aura effectué un tour complet sur lui-même.

Modélisation

L'idée, ici est de pourvoir réutiliser les informations récupérées pendant la phase d'apprentissage, afin de modéliser un espace 3D à partir des mesures effectuées pendant un déplacement du robot en direction des arêtes qui lui semblent les plus intéressantes.

Le modèle 3D se créerait au fur et à mesure que le robot explore la salle. Plusieurs techniques pourraient être utilisées et couplées au système pour avoir de bons résultats (flux optique, looming, etc…)

Bibliographie

1	Low overhead optical flow based robot navigation, Sebastien Dey (2003)
2	Visual looming as range sensor for mobile robots Developpoment of a visual object localization module for mobile robots
3	Deriving and matching image fingerprint sequences for mobile robot localization, Pierre Lamon (2000) Environmental modeling with fingerprint sequences for topological global localization, Pierre Lamon (2003)
4	Information Sampling for vision-based robot navigation. Niall Winters, José Santos-Victor
5	Projet movi, Modélisation, localisation, identification et reconnaissance pour la vision par ordinateur, INRIA (2002)
6	Good Features to Track, Jianbo Shi , Juin 2003
7	A Color Interest Operator for Landmark-based Navigation. Zachary Dodds and Gregory D. Hager. Department of Computer Science. Yale Univ 1997
8	Machine Vision, Ramesh Jain, Rangachar Kasturi, Brian G. Schunck

Annexe 1

Schéma de dépendance des package